DOI: 10.2507/29th.daaam.proceedings.035

A COMPARISON OF METHODS FOR THE IDENTIFICATION OF TRAFFIC LEVELS IN CITIES

Pavel Pokorny

This Publication has to be referred as: Pokorny, P[avel] (2018). A Comparison of Methods for the Identification of Traffic Levels in Cities, Proceedings of the 29th DAAAM International Symposium, pp.0243-0249, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-20-4, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/29th.daaam.proceedings.035

Abstract

The main task of this article is to describe the software solutions for the identification of live traffic levels in cities anywhere in the world. Two different methods are used and compared in order to apply these results in the industrial environment. Both methods are based on Google Maps solutions. The first method uses objects and methods directly in its programming interface (API), which can return the necessary values - that can then be processed. The second method uses a graphics representation of Google Maps with a traffic layer, where traffic levels are visualized using different colours. The required maps are then converted into a bitmap image where the colours are identified and then processed.

Keywords: identification; traffic level; map; image processing; programming

1. Introduction

Traffic on roads means the movement of road-users - including persons, streetcars, buses, vehicles, ridden or herded animals, and other conveyances; either singly or all together, while using the public right-of-way for travel purposes. Along with the increasing density of traffic, is the need to organize it. Traffic laws were defined for this reason. They define the collection of rules - with well-established priorities, rights-of-way, lanes, and traffic control at intersections.

Traffic estimation and prediction systems can offer solutions in order to improve traffic conditions and reduce travel delays by facilitating better utilisation of available capacity [1]. The success of traffic systems is heavily dependent on the availability of timely and accurate estimates of prevailing and emerging traffic conditions. When the correct information is available, different traffic prediction models and algorithms can be used [2] [3].

In order to attain fluent traffic flows - and safety, one can use live traffic level monitoring. The data obtained in this way can then be processed for evaluation purposes and then for the control of junctions or to calculate the current optimal routes between different places, (e.g. to avoid roads with very slow actual traffic flows - caused by a traffic accident, for instance [4]). Google Incorporated also offers certain solutions for these problems. It offers its own maps and supplies its own services to use them. These maps include satellite imagery, street maps, 360° panoramic views of streets (Street View), real-time traffic conditions (Google Traffic), and route planning for traveling by foot, car, bicycle (currently in beta-mode), or public transportation [5]. The Google Maps services include options that allow one to change the appearance of a selected map and to extend its visualization with different layers and filters.

One such is a traffic layer that can show live traffic-flow levels, mainly on roads and their intersections with high traffic-flow levels. The level is represented by different colours, and is regularly updated in a matter of minutes. The Google Maps services further offer their own programming interfaces (APIs), which can return information regarding distances or travel duration between different places [6]. With obtaining this information, the traffic levels can be then identified.

Both solutions can be used in the proprietary applications to process the information so obtained, and generate the requisite information for further use. This paper describes these applications and experiences obtained during the testing process. These applications are based on web technologies and can be run on any web browsers. The greatest advantage of this solution is therefore, free-of-charge availability - whenever and wherever there is an access to internet and for anyone who has a device that supports web-browsers (personal computers, laptops, tablets, smart phones, etc.).

2. Application Requirements

The created applications should be used in the industrial environment of the traffic-monitoring field. Therefore, they ought to meet the main specifications that real deployment requires. The following points describe these specifications.

- The first application that uses Google Maps APIs must include the following functionalities: the ability to read a configuration from a JSON file; the initialization of Google Maps APIs; periodic retrieval of travel times by calling Google Maps services for all points loaded from the configuration file; saving the requisite values into the output JSON structure [7]; and, sending them to a server application for further processing.
- The second application that uses graphics for the visualization of Google Maps must include the following functionalities: the ability to read a configuration from a JSON file; the initialization of Google Maps - with the added traffic layer; the periodic identification of traffic-flow levels for all points loaded from the configuration file; and, the determination of the values saved into the output JSON structure; which should be then sent to a server application for further processing.
- The input configuration file must have a JSON (JavaScript Object Notation) structure. This includes these items: a set period for the repetitive re-identification of a traffic value from Google Maps; the number of available traffic levels and their definition by colours in the RGB colour model [8]; a list of points on the maps - where such identification measures are then performed; and, a URL address where the JSON structure output is sent.
- The list of points contains the following properties for each point: latitude, longitude, a name, and an identification value.
- The output of the JSON structure must contain an array of points; where each of them includes latitude, longitude, name, identification value, and traffic-flow level.
- The traffic-flow level for the output must contain an identification value between 0 and 4. The value 0 represents an undefined traffic level; 1 - green colour, (fast traffic); 2 - orange colour, (slower traffic); 3 - red colour, (slow traffic); and 4 - dark red colour, (very slow traffic) on the map.
- The application that shows the selected map, should allow users to add their own visualization style. The user then can perform - or turn off, any map features - for example, landscapes, names of streets, map objects, etc.
- Both applications must be created in the JavaScript programming language [9] and have to be fully functional in all massively-used Web browsers (Chrome, Firefox, etc.). This means that the application can also be stored and run in a local server, or also be run on a client computer.
- The designed applications must also contain simple and clear programmer documentation. Its source code must be written with English syntax.
- Both applications should contain the following files: index.html (the main file that defines the user-interface and encapsulates all other files); con-fig.js (the input configuration file); and goots.js (this file contains a logic application). If necessary, it may also include other files, like css files [10], or pictures, etc.

3. Google Maps

Google Maps offers a mapping solution used by millions of websites and maps to provide experience for their users. In addition to manipulate with maps, they also perform through the programming interface working with Android, iOS, various web browsers and via HTTP protocol web services [11].

Access to the Google Maps resources is by means of an API key. This key is ob-tained, free-of-charge, after registering the software project on the Google API Console; a user can access up to 25,000 map-loads per 24-hours. Upon exceeding this free-usage limit, any further map-loads must be paid for [5].

The base object of Google Maps is the Map class. Using its constructor, it creates a Map Object. This class encapsulates lot of properties and methods - including which maps are shown, map-type, zoom and style. Most types of maps that Google Maps offers can be displayed; like for instance, road-maps, satellite-maps (displays satellite images), terrain (physical maps based on terrain information), and hybrid maps (mixture of roadmaps and satellite-views).

Fig. 1. Two examples of the same maps with different map-styles [12]

3.1. Google Maps APIs

The base object of Google Maps is the Map class. Using its constructor, it creates a Map Object. This class encapsulates lot of properties and methods - including which maps are shown, map-type, zoom and style. Most types of maps that Google Maps offers can be displayed; like for instance, road-maps, satellite-maps (displays satellite images), terrain (physical maps based on terrain information), and hybrid maps (mixture of roadmaps and satellite-views).

Using a styled map, a user can customize the presentation of the Google base-maps, by changing the visual display of elements like roads, businesses, parks - and other points-of-interest. The map-style is organized in a JSON-type structure [13].

This structure consists of three elements – featureType (geographic characteristics on the map - including parks, roads, bodies-of-water, and more); elementType (sub-parts of a feature - including geometry and labels); and stylers (indicate the visibility, colour, and weight of the feature). To specify a style, the user must combine all these three elements into a Style Array. Figure 1 shows two examples of the same maps with different styles [12].

Google Maps can also be modified to include transit, traffic or bicycling layers in order to display current traffic conditions, or bicycling-route and transit information. These layers are available in selected regions. The subject-of-interest in this paper is the traffic layer. Where supported, this layer adds real-time traffic information, to user maps by using the TrafficLayer object. The traffic information is then refreshed frequently - but not instantly. Rapid consecutive requests for the same area are un-likely to yield different results.

3.2. Google Maps APIs Services

Google Maps APIs also provide some useful services for calculating directions (efficient paths), travel distances, elevation data for locations on the surface of the earth, geocoding (converting addresses into geographic coordinates), different zoom levels for map type imagery and Google Street View Service for panoramic 360 degree views from designated roads throughout its coverage area [11].

For the purposes described in this paper, a travel distance service was used. This service, called Distance Matrix, can compute not only travel distance, but also journey duration among multiple origins and destinations using a given mode of travel [14].

Distance Matrix service input parameters contain following information: an origin array that contain one or more objects from which to calculate distance and time; an destination array that contain one or more objects to which to calculate distance and time; travel mode (bicycling, driving, walking, transit); transit option (required only for transit travel mode – here can be set arrival and departure time and transit mode (bus, rail, subway, train, tram)); and driving options (required only for transit travel mode – here can be set departure time and a traffic model - best, pessimistic or optimistic estimates).

After successful calling Distance Matrix service, user obtains an object, which contains all required information - distance and duration information for each origin/destination pair for which a route could be calculated and list of origin and destination addresses.

4. Application Solution

In order to identify traffic levels in cities, both applications only need HTML5 and JavaScript technologies [15] [16] to be developed. An appearance of the user interface is performed by using CSS language [6]. The concept was designed in order to attain the whole set of requirements described above in Chapter 2.

JavaScript is a high-level, dynamic, un-typed and interpreted run-time language [9]. Its functions are often called “using events” and “callback functions”. Created applications are based on functions that must be “called” in the correct order - which is performed by events or global variables.

4.1. Traffic Levels Identification Using Google Maps API

The application structure is contains following functions:

- *loadJsonFile* – this function performs the loading of the input configuration from the input JSON file, named config.js. It also stores this configuration into the global object.
- *saveJsonFile* – this function stores the output global JSON object into the external output file and also can sent it to a server application. This global object contains all required output information.
- init – the init function contains the commands for the initial operations in the application. This includes obtaining access to all of the elements on the web-page; calling the loadJsonFile function; the correct input data conversion; and further, calling the main function.
- *mainFunction* – this is the main function of the application. It controls whole operations in cycles for all points where the traffic level should be determined. Commands in this function decide if all points were not processed (then the processAllPoints function is called), or the processing process is finished - (the saveJsonFile function is called).
- *processAllPoints* – is a function that controls one cycle (in defined time) for all points where the traffic level should be determined. The algorithm calls the DistanceMatrixService function with all required input parameters described above (all origin and destination points and driving travel mode). Obtained values (distance and duration) is ten written into the web-browser console and saved into the output JSON object.

In addition, this application also includes a global object, which encapsulates global variables, constants, input and output JSON objects. When the application was finished, it was then tested in our home city Zlín and its vicinity for some hours. During this time, the processAllPoints function was periodically called every 5 minutes for 8 hours. When the application was finished, the output values were analysed.

The analysis unfortunately showed that the DistanceMatrixService function returns the same values of traffic duration in different time. So it seems to these values are not live-updated.

4.2. Traffic Levels Identification Using Image Processing

The structure of this application is similar to previous software. The only difference is in the content of processAllPoints function and two new functions were added:

- *processAllPoints* – is a function that controls one cycle for all points where the traffic level should be identified. The algorithm is developed in order to check if all previous points were processed - (then the processOnePoints function is called) - or not, (then this function waits).
- *processOnePoint* – this function processes one point where the traffic level should be determined. It performs the loading of the required Google Map, based on set parameters into the div element. The map style is also set here. In order to easily identify colours in a traffic layer, all unnecessary objects in the map are hidden. Further, after loading a map, processOnePoint calls the mapToCanvas function.
- *mapToCanvas* – this function is the main core of the whole application. It performs div-to-canvas conversion, the algorithm for traffic level identification, the storage of all of the required output values into the global JSON object, as well as the showing of these results in the web-browser console (used for testing).

Fig. 2. Example of a map with an added traffic layer

At first, these algorithms loads and shows a map with an activated traffic layer. An example of this map is shown in Figure 2. The colours of this layer display a traffic situation at the junction located near the theatre in (Central) Zlín - at noon; on 9.12.2017. Green colours represent fast traffic-flows and red represents slow traffic flows.

It is not possible to determine traffic levels in a defined place from shown Google Map, because the map div element on a web-page does not allow the colour identification of pixels. This problem can be solved via means of the canvas element - which represents a raster area where each pixel can be read, and its colour can be obtained. For these reasons, the attention was focused on possible ways to convert the map into a canvas element. Internet resource [17], showing that the html2canvas JavaScript library can solve this problem.

The html2canvas JavaScript library allows a user to capture "screenshots" of webpages or parts of them, directly on a web-browser. The screenshot is based on a Document Object Model (DOM), and as such, may not be 100% accurate to the real representation since it does not make an actual screenshot, but builds the screenshot based on the information available on the page [18]. A user can set some different parameters when using this library. These parameters include the size of the canvas, a url address to the proxy which is to be used for loading cross-origin images, the timeout for loading images, etc.

The traffic level is identified on the basis of pixels´ colours. JavaScript supports the getImageData method for canvas objects. getImageData returns a simple image object that holds colour values of every pixels inside a rectangle with defined sizes, on a canvas. For every pixel in the image-object, there are four pieces of information; the RGBA variables [4]. These variables can very easily then be compared with the values defined in the input configuration file - and the result determines the traffic level in the range of 1 – 4. If the obtained variables are not equal to any of the stored values, the traffic level is evaluated as undefined - and the output result then set to 0.

The only problem is to determine the exact pixel position where the current traffic level is required. If the application converts the whole map into a canvas with the same size, the “searched pixel” is located in the middle of the canvas. The geographical coordinates of the location where the level of traffic should be determined must be known very accurately. Every inaccuracies can cause bad positions on the map, and then bad identifications.

There also were several problems in developing this application solution. The first problem is that sometimes, the conversion library does not div to canvas conversion when the application is not on the screen (it runs in the background of the operation system). This error is created during the conversion process, because the div map element is always drawn correctly. This problem can also be created due to the conception of the div element, where the map is displayed. This element contains a lot of partial HTML elements - including links to Google Incorporated servers. Because the conversion process takes several seconds, these links may not be valid when they are connected again. This problem cannot be resolved in this application.

The second is that the traffic level layer may not always be visible on the map. Experience shows that this could be mainly at night. This problem occurs on the Google Maps server, and it is not also possible to correct it in this application. Finally, the traffic layer may not always be drawn in the same position. However, this shift is only in the order of several pixels. This problem can be solved with an evaluation of the neighbouring pixels. The program source-code can also be extended with a tolerance value that defines the permissible degree of colour difference, because the traffic lines are drawn with anti-aliasing.

5. Conclusion

This contribution describes two software solutions that identify live traffic-levels in cities. Both solutions - based on the JavaScript and HTML5 technologies are usable on all common web-browsers. Compared to other solutions, these technologies and implementations offer the following benefits. Anyone can obtain real-time information of traffic flows on the required place anytime. A person only needs any device that support web-browsers (personal computers, laptops, tablets, smart phones, etc.) and internet connections.

The first solution is based on the Google Maps APIs. It offers some services, which can return distances or travel duration among different places. The obtained values can then be processed in order to determine live-traffic levels. The created software tested these services in different selected places, but they returned the same values in different time. So it seems to these values are not live-updated.

The second software solution is based on the graphics representation of Google Maps with a traffic layer. It converts the selected maps into an image, identifies the colours of pixels - and from this, evaluates a traffic-level. However, there can sometimes be problems with displaying a traffic level and during the conversion process, as mentioned above. The described problems cannot be solved by user applications; so it is necessary to wait for an API that uses library updates or looks for another solution.

6. Acknowledgments

This application was created in collaboration with the Incinity Company, which develops complex integrated platforms for smart cities.

7. References

[1] Bozica, R. & Ante, R. (2015). Implementation of multi-criteria decision making in planning traffic systems, Proceedings of 26th DAAAM International Symposium on Intelligent Manufacturing and Automation, 21-25 October 2015, Zadar, ISSN 1726-9679, ISBN 978-390273407-5, Katalinic, B. (Ed.), pp. 69-74, Danube Adria Association for Automation and Manufacturing, DAAAM, Zadar, DOI 10.2507/26th.daaam.proceedings.010.

[2] Zeroual, A.; Messai, N.; Kechida, S. & Hamdi, F. (2018). A piecewise switched linear approach for traffic flow modeling. International Journal of Automation and Computing, Vol. 14, No. 6, December 2017, pp. 729-741, ISSN 1476-8186.

[3] Gao, I. & Liu, W. (2017). Cloud differential evolution algorithm for traffic flow prediction with least squares support vector machine, Proceedings of 2nd IEEE International Conference on Computational Intelligence and Applications, ICCIA 2017, 8-11 September 2017, Beijing, China, ISBN 978-153862030-4, pp. 189-193, Institute of Electrical and Electronics Engineers Inc., Beijing, DOI 10.1109/CIAPP.2017.8167205.
[4] Zovak, G.; Saric, Z. & Cop, A. (2011). Conceptual model for structuring traffic accident data, Proceedings of 22nd International DAAAM Symposium "Intelligent Manufacturing and Automation: Power of Knowledge and Creativity", 23-26 November 2011, Vienna, ISSN 1726-9679, ISBN 978-390150983-4, Katalinic, B. (Ed.), pp. 523-524, Danube Adria Association for Automation and Manufacturing, DAAAM, Vienna.
[5] https://maps.google.com, (2018). Google LLC, Google Maps, Accessed on: 2018-03-16.
[6] https://developers.google.com/maps/, (2018). Google LLC, Google Maps APIs, Accessed on: 2018-03-16.
[7] Smith, B. (2015). Beginning JSON, Apress, ISBN 978-1-484-0203-6, New York.
[8] http://learn.colorotate.org/, (2012). Colorotate, Fast and intuitive color editing, Accessed on: 2018-03-16.
[9] Stefanov, S. (2008). Object-Oriented JavaScript, Packt Publishing, ISBN 978-1-4302-4473-8, Birningham.
[10] Powers, D. (2012). Beginning CSS3, Apress, ISBN 978-1-4302-4473-8, New York.
[11] Svennerberg, G. (2010). Beginning Google Maps API 3, Apress, ISBN 978-1-4302-2802-8, New York.
[12] https://snazzymaps.com, (2018). Snazzy Maps, Free Styles for Google Maps, Accessed on: 2018-03-16.
[13] Rischpater, R. (2015). JavaScript JSON Cookbook, Packt Publishing, ISBN 978-1-78528-690-2, Birningham.
[14] https://developers.google.com/maps/documentation/javascript/distancematrix, (2018). Google LLC, Distance Matrix Service, Accessed on: 2018-03-16.
[15] https://w3.org/TR/html53, (2018). World Web Consortium, HTML 5.3, Accessed on: 2018-03-16.
[16] Pilgrim, M. (2010). HTML5: up and runnin, O´Reily, ISBN 978-0-596-80602-6, Sebastopol.
[17] https://stackoverflow.com, (2018). Stack Overflow, Where Developers Learn, Accessed on: 2018-03-16.
[18] https://html2canvas.hertzen.com/, (2017). Niklas von Hertzen, Html2Canvas – Screenshots with JavaScript, Accessed on: 2018-03-16.